



**Data Glacier**

Your Deep Learning Partner

# Web scraping in data science

Article writing

**Name:** Rania Tarek Fleifel

**Email:** [raniatarekfleifel@gmail.com](mailto:raniatarekfleifel@gmail.com)

**Country:** Egypt

**Specialization:** Data science

**Internship Batch:** LISUM13: 30

**Submission date:** 5<sup>th</sup> of November 2022

## Table of Contents

<b>Introduction.....</b>	<b>3</b>
<b>Section 1: Overview of web-scraping .....</b>	<b>4</b>
Definitions of web-scraping.....	4
Applications of web scraping .....	4
Types of web-scrapers .....	4
<b>Section 2: Pre-scraping .....</b>	<b>5</b>
What type of data is needed?.....	5
Where to obtain the data? .....	5
What are the storage resources available? .....	5
<b>Section 3: Technical stack in web-scraping context .....</b>	<b>6</b>
HTML .....	6
CSS .....	6
Devtool .....	7
BeautifulSoup .....	7
<b>Section 4: Permission to scrape .....</b>	<b>8</b>
How to check permission to a website's data? .....	8
<b>Section 5: Web-scraping tools .....</b>	<b>9</b>
Tool 1: HTTP requests .....	9
Retrieve from API.....	9
Retrieve from raw data.....	10
Tool 2 & 3: Selenium & Playwright .....	11
<b>Section 6: Steps of web-scraping through examples .....</b>	<b>13</b>
The basic steps of web-scraping .....	13
Example using HTTP requests .....	13
Example using selenium and playwright .....	16
<b>Section 7: web-scraping good practices .....</b>	<b>20</b>
<b>Conclusion .....</b>	<b>22</b>
<b>Appendix.....</b>	<b>22</b>
<b>References .....</b>	<b>23</b>

# Introduction

Availability of information is the enabler to the booming fields of data science and machine learning. Without an abundance of data, reaching conclusive insights and training predictive models wouldn't be possible. World wide web is our data goldmine. There's hardly any topic that no one has ever discussed somewhere on the internet or social media. While some of this data is available for download or through an API, a lot isn't. There's a lot of raw data that is unstructured, and this is where web scraping comes in to make data easily accessible for analysis and manipulation.

In this report we discuss web-scraping from data science point of view. This report is addressed to data scientists whose projects involves data that isn't readily structured for analysis, have errant data with missing values, or need to extract more features before creating their ML model. Although web-scraping includes RPAs and no-code solutions that automate the process of periodically retrieving data, we focus on building web-scrappers that combine open-source frameworks with python programming.

In the first section of this report, we present an overview of web-scraping. This discusses definition, distinction from web-crawling, types of web-scrappers, applications of web-scrappers. The following section focuses on factors that govern the choice of scraper depending on the project's needs. At the heart of web-scraping are HTML language, CSS styles, Devtools of browsers & parsing tools such as beautifulsoup. These factors are all discussed in the context of web-scraping in section 3. Before discussing web-scraping tools in section 5, namely http requests, selenium and playwright, we include a section that handles the ongoing struggle between website owners and web-scrappers. This section discusses the ethical dilemma that web-scraping introduces. It shows the topic from the perspective of both website owners, web scrappers and mentions each party's efforts in that regard. Section 6 presents the basic steps of web-scraping and implement them with the previously mentioned tools: http requests, selenium and playwright. Last but not least, we dedicate a section to web-scraping good practices that we've encountered and took note of while developing the examples in section 6.

# Section 1: Overview of web-scraping

There are several definitions of web-scraping, all of which focus on it being a process that extracts data from websites. The data is collected and then exported to a format accessible by users. Other names of web-scraping are web extracting or harvesting. While some sites use web scraping and web crawling interchangeably, I believe they have different end-results. Web-scraping targets a certain website to receive and parse data that is known before-hand. Web-crawling, on the other hand, visits a lot of sites, processes all the data it comes across, indexes it and saves it in some form of database. Embarking on a web crawling process when you need specific information from specific websites is a waste of resources. [1]

Applications of web scraping: When web-scraping is mentioned, the applications that come to mind all contain dynamic data. Assuming we have a company that is looking to launch a product, this company decides to apply a two-fold web-scraping project

1<sup>st</sup> fold price monitoring → This procedure is necessary to determine their pricing strategies. The company scans the competition websites and decide on the price that is in-tone with the rest of the market without jeopardizing their profit margin. [2]

2<sup>nd</sup> fold consumer sentiment analysis → By taking note of consumers reviews and comments about competing products, the company can focus on the deficiencies to have a clear edge over their competition.

There are various types of web-scrapers. If the personnel doing the web-scraping task doesn't have programming background, pre-built web scrapers are the way to go. This includes tailor-made browser-extension web scrapers or the more generic alternative Software web scrapers that is downloadable and compatible with various browsers. If there's a big budget specified for this task, it can be outsourced to a company that provides the scraper and an offsite cloud to run it, elevating the limitation of computer resources. In this report, we consider the web-scraping is done by someone who has programming background, can implement a Self-built scraper on his/her local machine. [3]

## Section 2: Pre-scraping

There are various factors that should be considered before proceeding to scrape data off the web. Clearly-defined requirements make way to making smart and efficient decisions regarding the tools used. Especially since web-scraping covers a range of tools that vary in their complexity and efficiency according to the project's needs. [4]

- a) What type of data is needed?
  - Static: A manual copy & paste, which is the easiest form of scraping, could suffice. A downloadable .csv though the website's API might do the job perfectly. A one-time scraping session where the data is processed post-scraping is standard-procedure for this type of data.
  - Dynamic: This is at the heart of web-scraping; dynamic data is the data that requires periodically-repeated scraping processes. This data can be further categorized into:
    - o Costumer-personalized data: YouTube and Netflix list of suggestion based on user's history on the website
    - o Data that are constantly updated (economic & societal indicators, ongoing season sports' dashboard)

**\*\* Disclaimer:** this is different from dynamic and static web-pages. A static web-page is one where the html source code matches what we see on the browser and a dynamic web-page is one where there's a mismatch. The mismatch is caused by the execution of java-script on the browser which we'll elaborate on in section tool#2&3 selenium and playwright.

- b) Where to obtain the data?
  - The website should be credible, up-to-date and neatly structured.
  - The website consents web-scraping of the data required. It's important to approach data ethically, and respect ownership and confidentiality.
- c) What are the storage resources available?
  - This factor is usually over-looked although it's crucial to the process. There's a compromise between the complexity of the scraper and the processing needed to be done on the data. If there's an abundance of storage space, using a very basic scraper that acquire a ton of data would suffice. Since the developer can sort through it locally. However, having limited storage requires using a scraper with features that pinpoints a more compact block of data that we can afford to save and work with.

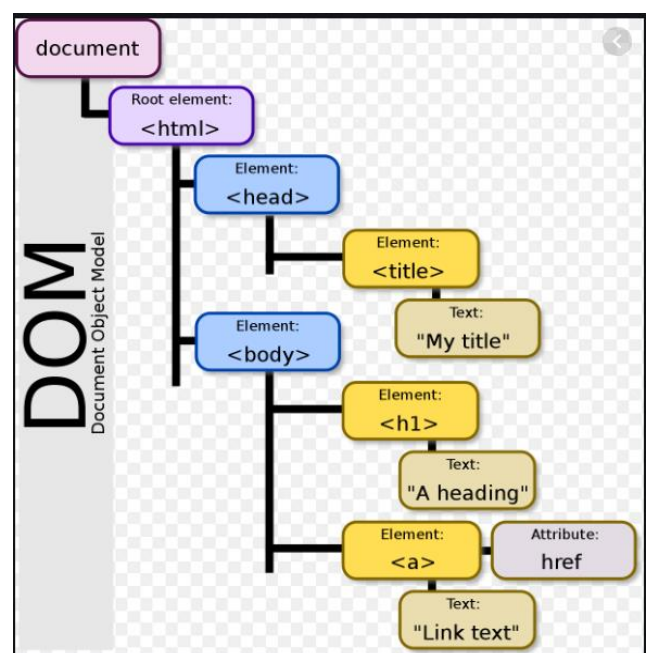
# Section 3: Technical stack in web-scraping context

The markup language that is used to create websites is called HTML. Unlike raw http, html allows formatting websites into a more inviting format. This is done through a series of tags. Most tags come in pairs that start with `<tag_name>` and ends with `</tag_name>` and encloses content and sometimes even attributes, among the important attributes is “id” which is heavily used since it uniquely identifies a certain tag. Tags can be easily nested as well. The following is a list of common tags:

- `<p>...</p>` to enclose a paragraph;
- `<br>` to set a line break;
- `<table>...</table>` to start a table block, inside; `<tr>...</tr>` is used for the rows; `<td>...</td>` cells;
- `<img>` for images;
- `<h1>...</h1>` to `<h6>...</h6>` for headers;
- `<div>...</div>` to indicate a “division” in an HTML document, basically used to group a set of elements;
- `<a>...</a>` for hyperlinks;
- `<ul>...</ul>`, `<ol>...</ol>` for unordered and ordered lists respectively; inside of these, `<li>...</li>` is used for each list item.

This figure shows the structure of an html page in a tree structure. This is actually what DOM interface does. It allows access to developers to the styling of an html through a database which is much more convenient and easier to follow and adjust.

Before we proceed to using the tags as pointers to the required data, let’s also briefly mention CSS that goes hand in hand with html styling. The best way to show how these two coexist, I quote []: “HTML is still used to define the general structure and semantics of a document, whereas CSS will govern how a document should be styled, or in other words, what it should look like.”



<https://iqss.github.io/dss-webscrape/concepts.html>

These styles are included in a document in different ways:

- 1) Inside a regular `<p></p>` html tag with “style” as an attribute. “`<p style='color: 'red';'>...</p>`”
- 2) Inside a `<style>.. </style>` tag placed in `<head>` tag of a page.
- 3) Inside a separate file which is referred to by a `<link>` tag in `<head>` tag. (Preferred)

As previously mentioned, we need some way to look through an html without rendering the whole page into python. For this function we resort to the Devtool of a browser [5]. This tool is available for all browsers, but I’m using chrome for this report. The devtool allows any user to use the “inspect” functionality which shows multiple tabs. We focus on “elements” tab that shows the source of the URL in a tree-based format. A very helpful option is that once you hover over a certain part of the elements tab, the corresponding section on the URL is highlighted. This is helpful because you can easily determine the tags related to the data you need to retrieve.

Now that we can locate the exact section where the data required is, we need a method to access this specific tag or class from python. This is where BeautifulSoup comes in. It’s a python library that does exactly what its name implies. It makes sense of soup -aka a mess of retrieved data-. You can access a specific part of a URL easily or find all instances where a certain tag appears save only those. Other methods that perform the same functionality are *regular expressions* and *lxml*. However, both approaches are originally written in C, which is beyond the scope of our report. [6]. Although selenium and playwright have parsing capabilities, we use BeautifulSoup, across all three for consistency. Also, according to [16] and [17], selenium’s parsing capabilities is sub-par to what’s available through BeautifulSoup and playwright’s is clunky and can break easily.

## Section 4: Permission to scrape:

The field of web-scraping is controversial -to say the least. There are opinions on both sides about its ethicality. The dispute against web-scraping is led by site-owners while web-scrapers, of course are in favor of web-scraping. Site-owners believe that any usage of data on their websites should be either with their explicit permission or not at all. Web-scrapers on the other side argue that public information is open-access and that rendering access to it cripples the advancement in all data-centric fields.

With courthouses having to deal with their share of cases related to data-harvesting, some of them breaching the confidentiality of site-users [7], new regulations attempted to govern this relationship. EU's GDPR (General Data Protection Regulation) and California's CCPA (California consumer privacy act) state that costumer's data that could "identify" them is off-limits unless explicit consent is given. [8][9]

Websites are becoming more explicit about their -how to use our data- permissions and they continue to add features to identify and slowdown -or permanently block- bots. Among those features are IP blockers and captcha blockers. On the other hand, web-scrapers continue to find ways to sur-pass those blockers.

So, how to check permission to a website's data?

- 1) read the terms and conditions of use of a website (can be mentioned explicitly, or listed during the sign-up check)
- 2) read the copyrights
- 3) read robots.txt which can be found @<http://<base-url>/robots.txt>. This file governs which type of user/bot has access to what data. [10]
- 4) Avoid scraping data that could identify a person (name, phone-number, address, ethnicity, etc.)
- 5) To sur-pass the blockers:
  - Use a proxy server and/or a dynamic IP address. To avoid getting your IP address blocked.[11][12]
  - Avoid sending too many requests to a website in a short time. If you have to send many requests, incorporate a sleeping time in your code that guarantees you're no longer banned [13]



# Section 5: Web-scraping tools

## Tool 1: HTTP requests

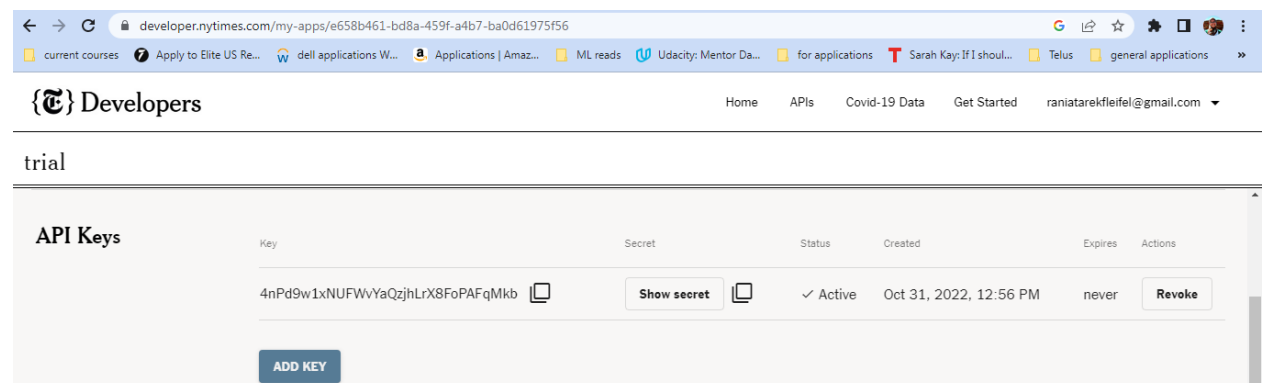
In this method, we make use of python library “Requests” specific for HTTP. This library is used to retrieve data in its raw form or from a site’s API [14]. Send an HTTP request to the URL required. We pass the URL we need to retrieve to *requests.get()* method. This method returns *requests.response()* object that contain information about the http provided.

Information returned from *r=requests.get()*:

- *r.status\_code* if *r.reason*  
200 if “ok”, 404 if “not found”, 429 if “too many requests”  
More about status codes, their reasons and what they mean can be found in [14].
- To access the textual form of the returned object, *r.text()* is used.
- *r.request()* and *r.header()* return information detailing the request fired toward the URL as well as the all http headers included in the response.

### A) Retrieve from API:

To use NYTimes API, you need to first create an account and request an API key to be able to retrieve its data.



This data is then incorporated in the code to access the data.

```
KEY = "4nPd9w1xNUFWvYaQzjhLrX8FoPAFqMkb"
date = datetime.datetime.now().strftime("%Y%m%d")

BASE_REQUEST = (
    "https://api.nytimes.com/svc/search/v2/articlesearch.json"
)

payload = {
    "api-key": KEY,
    "begin_date": date,
    "end_date": date,
    "q": "Donald Trump"
}

r = requests.get(BASE_REQUEST, params=payload)

if r.status_code == 200:
    print(r.json())
```

The returned data is in a dictionary form which require some further data parsing to narrow it down to the required info.

## B) Retrieve raw data:

There's no need for specific permissions. This returns a messy raw replication of the html page with all tags included. This is excessive since we need specific info and don't need to render the whole page's contents.

```
r = requests.get('https://en.wikipedia.org/wiki/Web_scraping')

if r.status_code == 200:
    print(r.text)

<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Web scraping - Wikipedia</title>
<script>document.documentElement.className="client-js";RLCONF={"wgBreakFrames":false,"wgSeparatorTransformTable":["",""],"wgDigitTransformTable":["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":["","January","February","March","April","May","June","July","August","September","October","November","December"],"wgRequestId":"00a16b4a-bdbd-4f0a-8ea4-3789a750b30d","wgCSPNonce":false,"wgCanonicalNamespace":"","wgCanonicalSpecialPageName":false,"wgNamespaceNumber":0,"wgPageName":"Web_scraping","wgTitle":"Web_scraping","wgCurRevisionId":1113541910,"wgRevisionId":1113541910,"wgArticleId":2696619,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"view","wgUserName":null,"wgUserGroups":["*"],"wgCategories":["CS1 Danish-language sources (da)","CS1 French-language sources (fr)","Articles with short description","Short description matches Wikidata","Articles needing addi
```

The messy text response we get, some might describe as “soup” where the data required is lost within so much un-needed data. This is where BeautifulSoup later comes in.

## Tool 2 & 3: Selenium & playwright

In the previous section, we discussed http requests and their usage in web-scraping. Let's recall that the `requests.get()` is the only instance where we are in contact with the website. So, what happens when there are elements of the URL that need more time to load? This is the case, with the dependency on Java in web contents. Java is built on top of http, meaning we now need to emulate the browser behavior to be able to run JavaScript and load its data. This is the idea of using web test automation tools such as selenium/playwright. These tools deal with data rendered using JS on websites. In other words, `requests.get()` deal with static content while automation tools that has rendering power deals with dynamic content.

We discuss two tools, namely Selenium and Playwright. Selenium is a the dominant more established (web browser wrapper) tool with a huge amount of open source contribution. It supports lots of programming languages like Java, c#, Python, Kotlin, Ruby, JavaScript and almost all browsers. It uses the webdriver API as means of interaction between web browsers and drivers. It translates test cases in json to the browser, and wait for the browser to execute them and send back its http response. However, Selenium is synchronous; while rendering the JS, all other activity on the program is blocked. This is why Selenium is considered on the slow spectrum in terms of processing and is not ideal in case of huge amounts of data. It's also important to consider the need to initiate a new webdriver component for each browser.

Playwright is among the newest additions as a web automation tool backed by Microsoft. It works with both html and java. Due to its relative newness, it still has a small footprint but it's spreading due to its speed compared to selenium. Playwright uses a web-socket connection that sends all information in one connection since it stays open throughout the test. It supports both synchronous and asynchronous mechanisms. This is crucial when dealing with single-paged Java-loaded pages, because with asynchronous mechanisms during this wait-time, another block of the program can run.

Criteria	Playwright	Selenium
Browser Support	Chromium, Firefox, and WebKit (note: Playwright tests <b>browser projects</b> , not stock browsers)	Chrome, Safari, Firefox, Opera, Edge, and IE
Language Support	Java, Python, .NET C#, TypeScript and JavaScript.	Java, Python, C#, Ruby, Perl, PHP, and JavaScript
Test Runner Frameworks Support	Jest/Jasmine, AVA, Mocha, and Vitest	Jest/Jasmine, Mocha, WebDriver IO, Protractor, TestNG, JUnit, and NUnit
Operating System Support	Windows, Mac OS and Linux	Windows, Mac OS, Linux and Solaris
Architecture	Headless browser with event-driven architecture	4-layer architecture (Selenium Client Library, JSON Wire Protocol, Browser Drivers and Browsers)
Integration with CI	Yes	Yes
Prerequisites	NodeJS	Selenium Bindings (for your language), Browser Drivers and Selenium Standalone Server
Real Device Support	Native mobile emulation (and <b>experimental real Android support</b> )	Real device clouds and remote servers
Community Support	Smaller but growing set of community resources	Large, established collection of documentation and support options
Open Source	Free and open source, backed by Microsoft	Free and open source, backed by large community

<https://applitools.com/blog/playwright-vs-selenium/>

# Section 6: Steps of web-scraping through examples

## The basic steps of web-scraping are:

- 1) Decide on the website to scrape the data
- 2) Check the permission to scrape the website
- 3) Locate the urls needed to grab the data
- 4) Pinpoint clear locators to the data
- 5) Use parsing library to grab the data
- 6) Locally save parsed data in a structured format such as csv or json

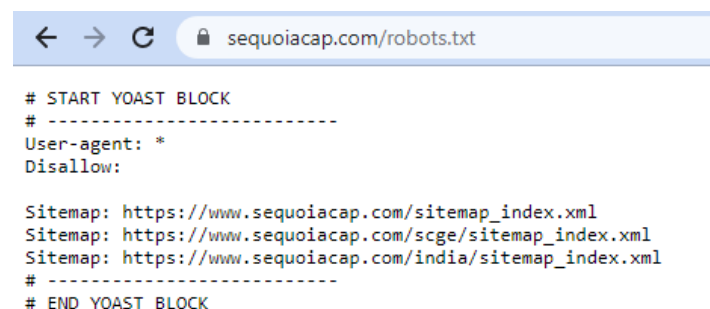
## Example using HTTP requests:

**\*\* Disclaimer:** the test case is used similar to that in [15], however we attempt to retrieve the data of the companies in the “spotlight” tab.

**Task:** scrape [sequoia website](#) – one of the biggest VC companies- for a list of companies consisting of:

- 1) Name of the company
- 2) Summary of the company
- 3) URL of the company
- 4) Social media accounts of the company
- 5) Description
- 6) Milestones
- 7) Team
- 8) Partners

**Permission:** Checking robots.txt file



```
# START YOAST BLOCK
# -----
User-agent: *
Disallow:

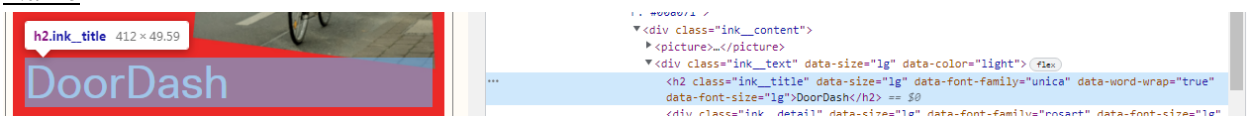
Sitemap: https://www.sequoiacap.com/sitemap_index.xml
Sitemap: https://www.sequoiacap.com/scge/sitemap_index.xml
Sitemap: https://www.sequoiacap.com/india/sitemap_index.xml
# -----
# END YOAST BLOCK
```

The \* in user-agent is a wildcard that means everyone is allowed. The file also specifically mentions three URLs that are not allowed. This means all else is allowed. We're interested in accessing <https://www.sequoiacap.com/our-companies/> so we're good for scraping.

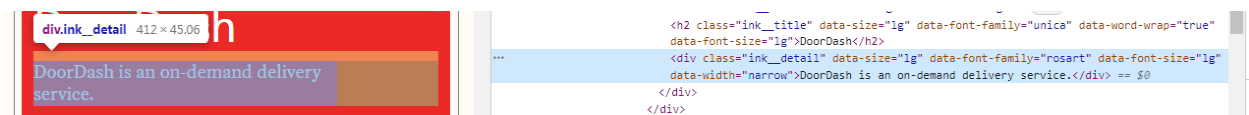
## Locating urls and data:

- Name, summary → <https://www.sequoiacap.com/our-companies/>
- URL, Social media accounts, Description, Milestones, Team, Partner of the company  
→ [https://www.sequoiacap.com/our-companies/<company\\_name>](https://www.sequoiacap.com/our-companies/<company_name>)

### Name

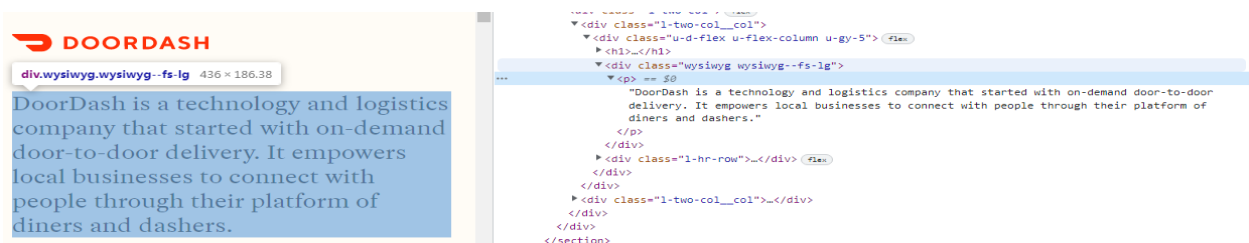


### Summary:



The rest of the info related to each company can be found in the url: [url+<company\\_name>](url+<company_name>)

### Description:



### URL:



### Social media accounts:



### Milestones, Team and Partners:

- The 3 points have identical tags and classes, except for the *h2.clist\_title.text* which we use for efficient coding
- Team had */t /n* for this styling so we stripped the string from all residuals.
- Partners has links to the partners, if the link isn't available the partner's name is mentioned.
- Also, if there's more than one partner, the *h2.clist\_title.text* reads "Partners", otherwise it reads "Partner"



### The saved .csv looks like this:

In [6]: `display(df_read)`

	name	summary	url	socialmedia	description	milestones	team
0	DoorDash	DoorDash is an on-demand delivery service.	<a href="http://www.doordash.com">http://www.doordash.com</a>	[ <a href="https://www.twitter.com/doordash">https://www.twitter.com/doordash</a> , <a href="https://www.facebook.com/doordash">https://www.facebook.com/doordash</a> ]	DoorDash is a technology and logistics company...	['Founded2013', 'Partnered2014', 'IPO2020']	[ <a href="https://www.sequoiacap.com/founder/andy-fang/">https://www.sequoiacap.com/founder/andy-fang/</a> ]
1	Nubank	Nubank provides next-generation mobile banking...	<a href="http://www.nubank.com.br">http://www.nubank.com.br</a>	[ <a href="https://www.twitter.com/nubankbrasil">https://www.twitter.com/nubankbrasil</a> , <a href="https://www.facebook.com/nubankbrasil">https://www.facebook.com/nubankbrasil</a> ]	Nubank is the leading financial technology company...	['Founded2013', 'Partnered2013']	[ <a href="https://www.sequoiacap.com/founder/david-veloso/">https://www.sequoiacap.com/founder/david-veloso/</a> ]
2	Zoom	Zoom brings teams together with cloud-based video...	<a href="https://zoom.us/">https://zoom.us/</a>	[ <a href="https://www.twitter.com/zoom_us?lang=en">https://www.twitter.com/zoom_us?lang=en</a> , <a href="https://www.facebook.com/zoom">https://www.facebook.com/zoom</a> ]	Zoom provides cloud video conferencing, online...	['Founded2011', 'Partnered2016', 'IPO2019']	[ <a href="https://www.sequoiacap.com/founder/eric-yuan/">https://www.sequoiacap.com/founder/eric-yuan/</a> ] [ <a href="https://www.sequoiacap.com/founder/eric-yuan/">https://www.sequoiacap.com/founder/eric-yuan/</a> ]
	Snowflake	Snowflake is			Snowflake		

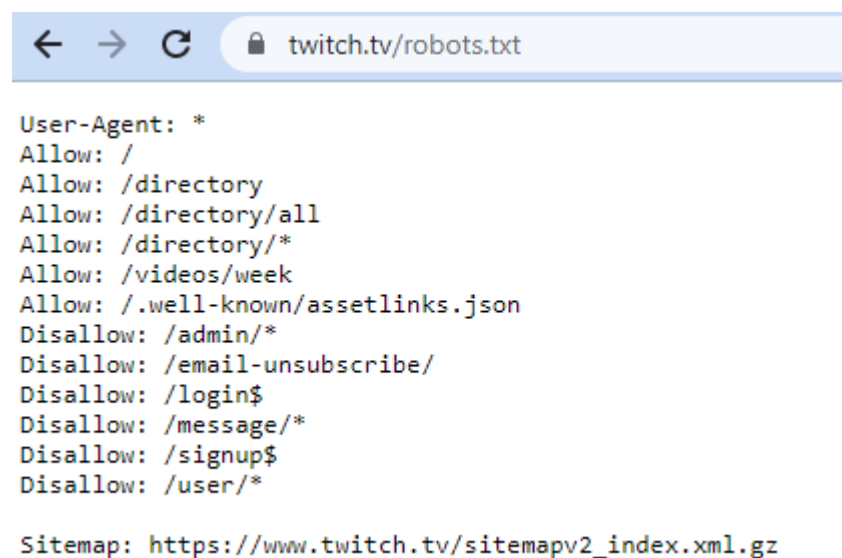
## Example using selenium and playwright:

**\*\* Disclaimer:** the test case is used in [16] and [17]. The point is to showcase how retrieving data from dynamic websites can be done in selenium and playwright.

**Task:** Scrape [twitch.tv](https://www.twitch.tv/directory/game/Art) website directory [directory/game/Art](https://www.twitch.tv/directory/game/Art). In this directory, users showcase their artistic process. The data to-be-scraped is:

- 1) title of the video
- 2) url to add to base url [<https://www.twitch.tv/>]
- 3) tags on the video
- 4) number of views

**Permission:** Checking robots.txt file

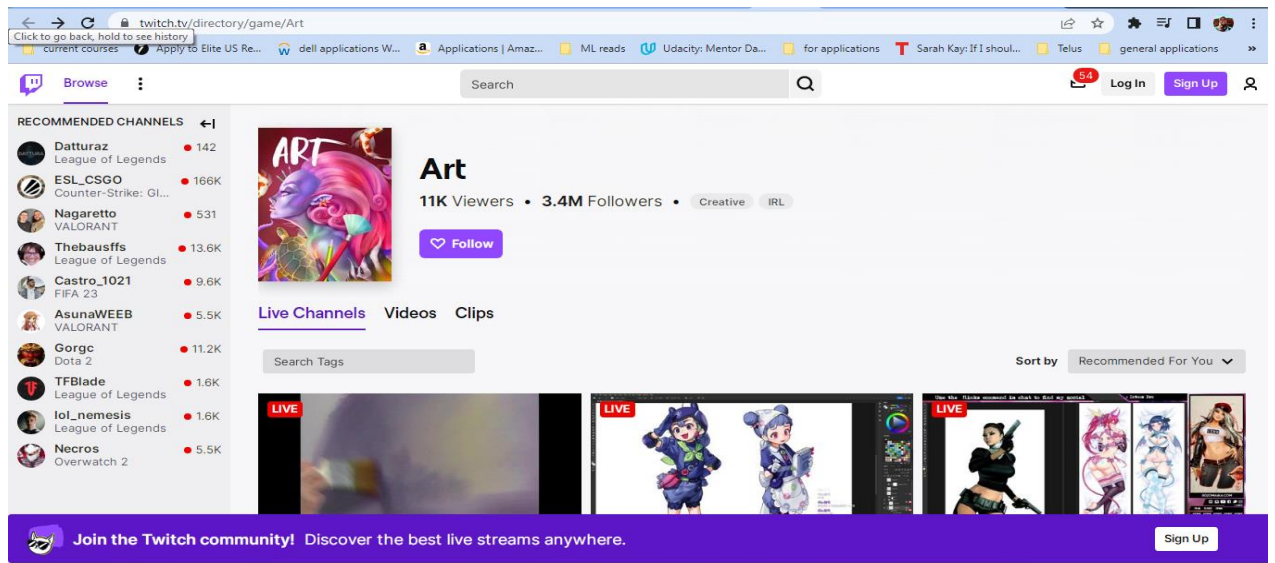


Before the \* in user-agent is specific permissions to different bots. For the rest of the user-agents, although it's not explicitly mentioned, accessing <https://www.twitch.tv/directory/game/Art> fall under the allowed `/directory/*`, so we're good for scraping.

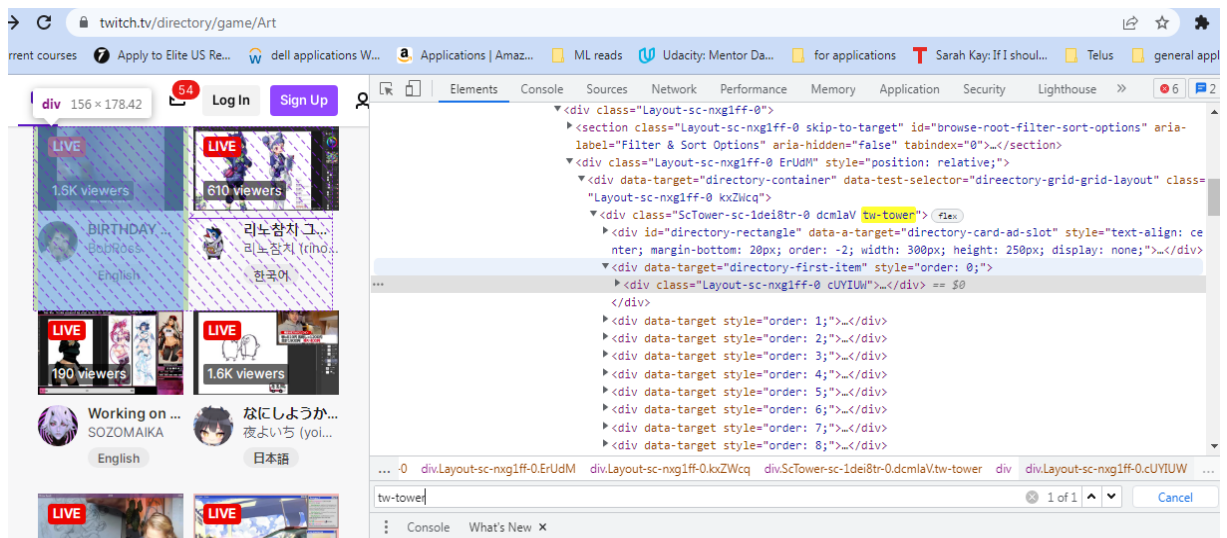


## Locating urls and data:

- All data required is available on the page

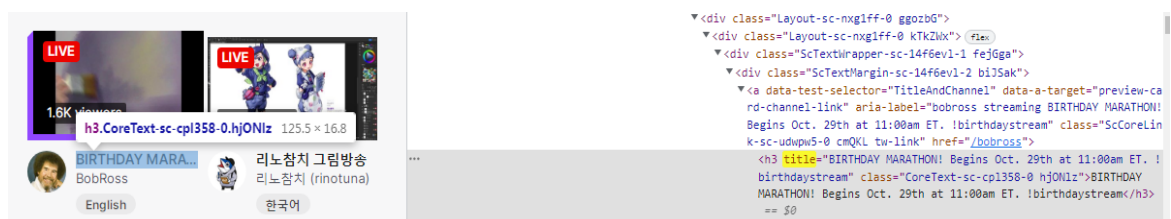


On further inspection, it's shown that all the required data is found inside `<div class='tw-tower'>`, `<div class='data-target'>`



Identifiers to retrieve each item of the data:

- title: `<h3 title`



- url: `<tw-link href`

```
...
<a data-test-selector="TitleAndChannel" data-a-target="preview-card-channel-link" aria-label="bobross streaming BIRTHDAY MARATHON! Begins Oct. 29th at 11:00am ET. |birthdaystream" class="ScCoreLink-sc-udwpw5-0 cmQKL tw-link" href="/bobross"> == $0

```

- tags: `<tw-tag::text`

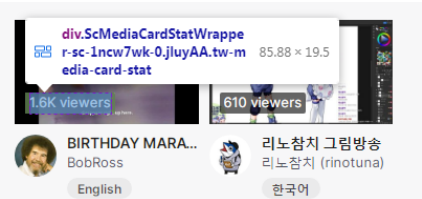


```

<div class="Layout-sc-nxglff-0 ggozbG">
  <div class="Layout-sc-nxglff-0 kTKZWk">
    <div class="ScTextWrapper-sc-14f6ev1-1 fejGga">
      <div class="ScTextMargin-sc-14f6ev1-2 bi7Sak">...</div>
    <div class="Layout-sc-nxglff-0 dRKpYM">
      <div class="InjectLayout-sc-588ddc-0 eXlw00">
        <div class="InjectLayout-sc-588ddc-0 beXCOC">
          <button class="ScTag-sc-xzp4i-0 iKNvdP tw-tag" aria-describedBy="BppEjMwKoAIFp2GrHcmQu200HRqJiqg" aria-label="English" data-a-target="English">
            <div class="ScTagContent-sc-xzp4i-1 gOMNvj">...</div>
          </button>
        </div>
      </div>
    </div>
  </div>
</div>

```

- number of views: `<tw-media-card-stat::text`



```

...
<div class="ScMediaCardStatWrapper-sc-1ncw7wk-0 jIuyAA tw-media-card-stat">
  <div class="ScPositionCorner-sc-1i1ybo2-1 gtpTmt">...</div>
  <div class="ScPositionCorner-sc-1i1ybo2-1 eHqCXd">
    <div class="ScMediaCardStatWrapper-sc-1ncw7wk-0 jIuyAA tw-media-card-stat">1.6K viewers</div>
  </div>
</div>
</div>

```

The saved .csv looks like this:

```
df_read
```

Out[10]:

	title	url	tags	viewers
0	BIRTHDAY MARATHON! Begins Oct. 29th at 11:00am...	/bobross	['English']	16
1	리노참치 그림방송	/rinotuna	['한국어']	841
2	☐ Cat mecha ☐   First mecha commission   Sketc...	/tofusenshi	['czech', 'artcommission']	229
3	Working on covers...   Istore llinks	/sozomaika	['English']	234
4	[Vtuber ITA] ~ SUBATHON di Compleanno! Day 7...	/cappuccino_chan	['italiano', 'art', 'illustration']	64
5	☀️ your daily dose of motivation ☀️	/vivisartservice	['English', 'art', 'positive']	93
6	art strim ( "ω" )	/ototaaan	['chill', '日本語', 'illustrator']	32

\*\* Post-script: This result shows the output of the selenium scraping. However, when I tried using similar code to retrieve the same content using playwright, there were errors that I looked through all types of resources and couldn't figure out how to overcome. If anything, this proves the bottlenecks that comes with using tools with little or no open-source contribution. The code is found in the appendix for reference.

The true edge of java rendering web-automation tools is its ability to emulate how humans interact with websites, this includes checking the captcha check, entering login username and password then either pressing physical “ENTER” key or the “Login” button on the page. It allows you to let the website load more content when you reach the end of the current page

and so much more. In the following example, we look at scraping data like we did with the HTTP requests example, but for more advanced functionalities, the following articles [18], [16] provide step-by-step to using selenium and playwright for much more than just basic scraping.

#### Post-scraping:

Supposedly that at this point, you have successfully cleaned your soup and pinpointed the data in-question. Now you just save the data you need on your local machine in a structured format such as csv or json. This way you can proceed with your project.

## Section 7: web scraping good practices

- 1) Scrape only the data you truly need. Scraping a lot of information is a waste of scraping resources, memory to save the scraped data and processing capabilities required to filter a pile of data to reach the required data.
- 2) Be mindful about loading the servers of the website you're scraping
  - a) If you're at a point where you figured way a cleaner expression to scrape your data, or cleaning your previous scrape → do not scrape again and just perform data-cleaning
  - b) Try to scrape websites during off-peak hours of their operation to avoid overcrowding the site's server.
- 3) Incorporate how unreliable web scraping is in your code. Web scraping can fail due to several reasons from connection errors to server errors. In [13] the developer attempts to access a file 3 times – to overcome any unexpected failure in the scraping process- and each time he attempts the scrape again, he increases the sleep time to provide enough time so that the server banning period is over.
- 4) If you have the option, don't opt for websites with Dynamic content. This type of pages is harder to parse. An example is Netflix accounts that is personalized according to users' preferences. [19]
- 5) Make sure the website has consistent styling throughout its pages. Some websites use this method to make it harder for bots to crawl them, others just don't care. You need to feed the web-scraper pointers to the data you need, whether it's a certain section or a font style. For instance, you provide a name of a table that you want the scraper to obtain from each year's page. If the table is named differently or styled differently in some years' pages, the scraper won't recognize the data.
- 6) Account for websites' changes, don't heavily rely on exact wording where a white space can mess up your scraper. One added line will change the styling, instead attempt to find your data with a unique identifier (e.g. id).
- 7) Consistency of styling in a website includes how hierarchical their links are. In [20], the site explains that each series has a unique identifier that can be easily saved and used periodically to check how a certain economic indicator is changing.
- 8) In [21] if you need to analyze stats of all NBA players, you can access the data easily by iterating on the alphabet which will point you to <https://www.basketball->

[reference.com/players/<your\\_current\\_letter>](https://www.basketball-reference.com/players/<your_current_letter>) where the link that leads to each player's stats can be accessed as [https://www.basketball-reference.com/players/<your\\_current\\_letter>/<4lettersoflastname><2lettersoffirstname><number>.html](https://www.basketball-reference.com/players/<your_current_letter>/<4lettersoflastname><2lettersoffirstname><number>.html)

Example: <https://www.basketball-reference.com/players/b/bryanko01.html> access data of player Kobe Bryant.

# Conclusion:

A lot of the projects beginner data scientists use to learn about the components of an end-to-end implementation provide a ready-made .csv or .json file to work on. This report handles the phase where this file is acquired. Web-scraping is a well-developed field of study that incorporates a lot of underlying information about how websites are developed and how users can interact with these websites. With the increasing popularity of automating redundant tasks, web-scrapers have become more intelligent and handles all the under-lying complications of websites. However, its important for data scientists to have enough knowledge to choose web-scrapers efficiently.

This report attempts to explain what we believe is the required technical stack that allows data scientists to make an informed decision about the type and complexity of web-scraaper needed for their projects. The hardest part of web-scraping is deciding which scraper to use. This is why we discuss the ever-going battle between data-scrapers and website owners. We provide steps to check the website's permission to access their data and good practices to avoid websites blockers. We discuss http requests, selenium and playwright methods to scrape static and dynamic websites. We use Python's BeautifulSoup to parse through the acquired data. In order to include the practical point-of-view of web-scraping. We implement examples using the three scraping methods afore-mentioned, and include good web-scraping practices and insights we used in the implementation.

# Appendix:

- Code used during this report can be found at the following link: [Code](#)
- Resulting .csv file from the http-request example:  
[webscraping\\_requests\\_output.csv](#)
- Resulting .csv file from the selenium/playwright example:  
[webscraping\\_selenium\\_output.csv](#)

# References:

- [1] C. Miranda, “What is Web Scraping in Data Science? | ParseHub,” 02-Apr-2021. [Online]. Available: <https://www.parsehub.com/blog/web-scraping-in-data-science/>. [Accessed: 05-Nov-2022].
- [2] C. Dilmegani, “Top 18 Web Scraping Applications & Use Cases,” 26-Dec-2020. [Online]. Available: <https://research.aimultiple.com/web-scraping-applications/>. [Accessed: 05-Nov-2022].
- [3] “What is Web Scraping and How to Use It? - GeeksforGeeks.” [Online]. Available: <https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/>. [Accessed: 05-Nov-2022].
- [4] “4 Things to Consider Before Starting Your Web Scraping Project | by Dan Suciu | Medium.” [Online]. Available: <https://dan-suciu.medium.com/4-things-to-consider-before-starting-your-web-scraping-project-a1f57ee381dc>. [Accessed: 05-Nov-2022].
- [5] “Chrome DevTools - Chrome Developers.” [Online]. Available: <https://developer.chrome.com/docs/devtools/>. [Accessed: 05-Nov-2022].
- [6] “2 Web Scraping Approaches | Web Scraping Using Selenium Python.” [Online]. Available: <https://iqss.github.io/dss-webscrape/web-scraping-approaches.html>. [Accessed: 05-Nov-2022].
- [7] “An update on scraping,” 06-May-2022. [Online]. Available: <https://news.linkedin.com/2022/may/an-update-on-scraping>. [Accessed: 05-Nov-2022].
- [8] “California Consumer Privacy Act (CCPA): What you need to know to be compliant | CSO Online.” [Online]. Available: <https://www.csoonline.com/article/3292578/california-consumer-privacy-act-what-you-need-to-know-to-be-compliant.html>. [Accessed: 05-Nov-2022].
- [9] “What is GDPR, the EU’s new data protection law? - GDPR.eu.” [Online]. Available: <https://gdpr.eu/what-is-gdpr/>. [Accessed: 05-Nov-2022].
- [10] “How to Check if a Website Allows Scraping?” [Online]. Available: <https://scrape.do/blog/how-to-check-if-a-website-allows-scraping>. [Accessed: 05-Nov-2022].
- [11] “Web Crawler Whitepaper.” [Online]. Available: <https://aimultiple.com/whitepapers/web-crawling>. [Accessed: 05-Nov-2022].
- [12] “Using Proxies for Web Scraping: How-to, Types, Best Practices.” [Online]. Available: <https://research.aimultiple.com/proxy-scraping/>. [Accessed: 05-Nov-2022].

- [13] “Web Scraping NBA Games With Python [Full Walkthrough W/Code] - YouTube.” [Online]. Available: <https://www.youtube.com/watch?v=o6Ih934hADU>. [Accessed: 05-Nov-2022].
- [14] “HTTP response status codes - HTTP | MDN.” [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status?retiredLocale=de>. [Accessed: 05-Nov-2022].
- [15] “Web Scraping 101 in Python with Requests & BeautifulSoup.” [Online]. Available: <https://www.statworx.com/en/content-hub/blog/web-scraping-101-in-python-with-requests-beautifulsoup/>. [Accessed: 05-Nov-2022].
- [16] “Web Scraping with Selenium and Python.” [Online]. Available: <https://scrapfly.io/blog/web-scraping-with-selenium-and-python/#parsing-dynamic-data>. [Accessed: 05-Nov-2022].
- [17] “Web Scraping with Playwright and Python.” [Online]. Available: <https://scrapfly.io/blog/web-scraping-with-playwright-and-python/#tip-playwright-in-repl>. [Accessed: 05-Nov-2022].
- [18] “Web Scraping using Selenium and Python | ScrapingBee.” [Online]. Available: <https://www.scrapingbee.com/blog/selenium-python/#full-example>. [Accessed: 05-Nov-2022].
- [19] “Beginner’s Guide for Web Scraping: Challenges & Best Practices.” [Online]. Available: <https://research.aimultiple.com/web-scraping-challenges/>. [Accessed: 05-Nov-2022].
- [20] “How to search | BPstat.” [Online]. Available: <https://bpstat.bportugal.pt/como-pesquisar>. [Accessed: 05-Nov-2022].
- [21] “Basketball Statistics & History of Every Team & NBA and WNBA Players | Basketball-Reference.com.” [Online]. Available: <https://www.basketball-reference.com/>. [Accessed: 05-Nov-2022].